MinimaLT: Minimal-latency Networking Through Better Security

W. Michael Petullo
United States Military Academy
West Point, New York USA
mike@flyn.org

Xu Zhang
University of Illinois at Chicago
Chicago, Illinois USA
xzhang@cs.uic.edu

Jon A. Solworth
University of Illinois at Chicago
Chicago, Illinois USA
solworth@ethos-os.org

Daniel J. Bernstein
University of Illinois at Chicago — TU Eindhoven
Chicago, Illinois USA — Eindhoven, Netherlands
djb@cr.yp.to

Tanja Lange
TU Eindhoven
Eindhoven, Netherlands
tanja@hyperelliptic.org

ABSTRACT

Minimal Latency Tunneling (MinimaLT) is a new network protocol that provides ubiquitous encryption for maximal confidentiality, including protecting packet headers. MinimaLT provides server and user authentication, extensive Denial-of-Service protections, privacy-preserving IP mobility, and fast key erasure. We describe the protocol, demonstrate its performance relative to TLS and unencrypted TCP/IP, and analyze its protections, including its resilience against DoS attacks. By exploiting the properties of its cryptographic protections, MinimaLT is able to eliminate three-way handshakes and thus create connections faster than unencrypted TCP/IP.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Protocol architecture

Keywords

Network security; protocol; encryption; authentication

1 Introduction

Our goal is to protect *all* networking against eavesdropping, modification, and, to the extent possible, Denial of Service (DoS). To achieve this goal, networking must protect privacy, provide strong (i.e., cryptographic) authentication of both servers and users, be easy to configure, and perform well. These needs are not met by existing protocols.

Hardware and software improvements have eliminated historical cryptographic performance bottlenecks. Now, strong symmetric encryption can be performed on a single CPU core at Gb/s rates [40], even on resource-constrained mobile devices [8]. Public-key cryptography, once so agonizingly slow that systems would try to simulate it with symmetric key cryptography [36], is now performed at tens of thousands of operations per second on commodity CPUs. Due to these advances, along with the threats found on the Internet, researchers are increasingly calling for the protection of all network traffic [10, 51, 20, 13].

However, one performance metric is a fundamental limitation—network latency [28]. Latency is critical for users [55]. For example, Google found that a latency increase of 500ms resulted in a 25% dropoff in page searches, and studies have shown that user experience degrades at latencies as low as 100ms [12]. In response, there have been several efforts to reduce latency for both TCP and encrypted networking [14, 41, 37, 10, 48, 56].

We describe here MinimaLT, a secure network protocol which delivers protected data on the first packet of a typical client-server connection. MinimaLT provides cryptographic authentication of servers and users; encryption of communication; simplicity of protocol, implementation, and configuration; clean IP-address mobility; and DoS protections.

MinimaLT's design intentionally crosses network layers for two reasons. First, security problems often occur in the seams between layers. For example, Transport Layer Security (TLS) is vulnerable to attacks on TCP/IP headers due to its layering; connection-reset (RST) and sequence-number attacks interrupt TLS connections in a way that is difficult to correct or even detect [18, 3, 59]. Second, multi-layer design enables MinimaLT to improve performance.

Particularly challenging has been to provide key erasure ("forward secrecy") at low latency. **Key erasure** means that even an attacker who captures network traffic and later obtains all long-term private keys cannot decrypt past packets or identify the parties involved in communication. Traditionally, key erasure is implemented with Diffie-Hellman key exchange (DH) in a way that imposes a round trip before sending any sensitive data. MinimaLT eliminates this round trip, instead obtaining the server's ephemeral key during a directory service lookup (§3.4 and §5.2). Furthermore, MinimaLT inverts the normal mandatory start-of-connection handshake, instead ensuring connection liveness using a server-initiated handshake only when a host's resources become low (§5.7). Eliminating these round trips makes MinimaLT *faster than unencrypted TCP/IP at establishing connections.*

A second challenge is to make connections portable across IP addresses to better support mobile computing. MinimaLT allows the user to start a connection from home, travel to work, and continue to use that connection. This avoids application recovery overhead and lost work for operations which would otherwise be interrupted by a move. MinimaLT IP mobility does not require intermediary hosts or redirects, enabling it to integrate cleanly into protocol processing (§5.6). To provide better privacy, MinimaLT blinds third parties to IP-address portability, preventing them from linking a connection across different IP addresses.

Public domain. This work was supported by the US National Science Foundation under grants CNS-0964575 and 1018836, the European Commission under Contract ICT-2007-216676 ECRYPT II and INFSO-ICT-284833 PUFFIN, and by the Netherlands Organisation for Scientific Research (NWO) under grant 639.073.005. Permanent ID of this document: 4613afc97aa0c7b8c45bc08f46c2e120. Date: October 31, 2013.
CCS'13, November 4–8, 2013, Berlin, Germany.
ACM 978-1-4503-2477-9/13/11. http://dx.doi.org/10.1145/2508859.2516737.

A third challenge is DoS prevention. A single host cannot thwart an attacker with overwhelming resources [30], but MinimaLT protects against attackers with fewer resources. In particular, MinimaLT dynamically increases the ratio of client (i.e., attacker) to server resources needed for a successful attack. MinimaLT employs a variety of defenses to protect against DoS attacks (§7.6).

A fourth challenge is authentication and authorization. Experience indicates that network-based password authentication is fraught with security problems [29, 54, 42, 11], and thus cryptographic authentication is needed. Our authentication framework supports both identified and non-identified (pseudonym) users (§3.1). We designed MinimaLT to integrate into systems with strong authorization controls.

To meet these challenges, we have produced a clean-slate design, starting from User Datagram Protocol (UDP), by concurrently considering multiple network layers. We found an unexpected synergy between speed and security. The reason that the Internet uses higher-latency protocols is that, historically, low-latency protocols such as T/TCP have allowed such severe attacks [14] as to make them undeployable. It turns out that providing strong authentication elsewhere in the protocol stops all such attacks without adding latency.

MinimaLT was designed for, and implemented in, Ethos, an experimental Operating System (OS) [52]. Ethos's primary goal is to make it easier to write robust applications, i.e., applications able to withstand attacks. MinimaLT serves as Ethos' native network protocol; it is part of Ethos' authentication suite which includes a Public Key Infrastructure (PKI) named sayI [53] and a networking API [47, 46]. We have also ported MinimaLT to Linux.

MinimaLT provides the features of TCP/IP (reliability, flow control, and congestion control), and adds encryption, authentication, clean IP mobility, and DoS protections, all while providing key erasure and reducing latency.

§2 summarizes the threats for which MinimaLT provides countermeasures. The next four sections explain how MinimaLT works and how fast it is: §3 describes the central objects, §4 describes the message format, §5 describes the protocol dynamics, and §6 provides a performance evaluation. §7 explains why MinimaLT was designed the way it was, and §8 compares MinimaLT to related work.

2 Threat model

We are concerned with an attacker that will attempt to violate the confidentiality and integrity of network traffic. Our attacker can observe and modify arbitrary packets, might be a Man-in-the-Middle (MitM), might provide fraudulent services which attempt to masquerade as legitimate services, or might attempt to fraudulently assume the identity of a legitimate user. An attacker who gains complete control over clients and servers, through physical access or otherwise, might be able to decrypt or identify the parties communicating with very recent and future packets. However, he should be unable to do so with older packets.

In addition, we want to weaken attacks on availability. DoS attacks from *known* users are expected to be addressed through de-authorizing abusive users or non-technical means. An *anonymous* attacker might try to affect availability, through transmission-, computation-, and memory-based DoS. An attacker with enough resources (or control over the network) can always affect availability, so we attempt to drive up his costs by making his attack at least as expensive as the cost to defend against it. Here the ability to spoof the source IP address of a packet *and* capture a reply should not allow much easier attacks.

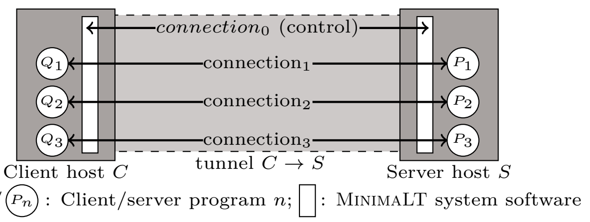


Figure 1: A tunnel encapsulates all of the connections between a given pair of hosts; connections are user-process-to-service and the server cryptographically authenticates each client user (the control connection does not involve application programs)

We are also want to make it difficult for an attacker to link a user's connection across different IP addresses. For example, if a user suspends his laptop at home and wakes it up at a wireless café, a network eavesdropper should not be able to infer from the protocol that these belong to the same user. We do not address the linking of flows associated with the same IP address, as this type of protection is better afforded by techniques such as onion routing [16].

3 MinimaLT objects

This section introduces the central objects that interact in the MinimaLT architecture. MinimaLT uses public keys to identify servers and users; creates encrypted tunnels between hosts, through which authenticated user-to-service connections are multiplexed; and publishes ephemeral server keys in a directory service to reduce latency by eliminating setup handshakes. To provide a complete picture for a MinimaLT deployment, this section concludes by describing how to integrate MinimaLT's directory service with DNS and the X.509 PKI (used by web browsers). This is not part of the protocol; it is merely one possible way to combine a directory service with the existing Internet infrastructure.

3.1 Public keys

MinimaLT is decidedly public-key-based. Both servers and users are identified by their public keys; such keys serve as a Universally Unique ID (UUID) [61, 50, 35]. Users prove their identity to servers by providing ciphertext which depends on both their and the server's keys (§5.8). A user may be known—i.e., the underlying system is aware of a real-world identity associated with the user's public key—or he may be a **stranger**—a user whose real world identity is unknown. We consider a stranger who produces a new identity for each request **anonymous**. Whether strangers or anonymous users are allowed is left to the underlying system's authorization policy.

3.2 Tunnels

A MinimaLT **tunnel** is a point-to-point entity that encapsulates the set of connections between a client and an authenticated server, as we depict in Figure 1. MinimaLT creates a tunnel on demand to service a local application's outgoing connection request or in response to the first packet received from a host (both are subject to the underlying system's authorization controls). In Figure 1, let Q_1 be the first program on client C to request a connection to server S. C first establishes a tunnel endpoint, and then sends a mes-

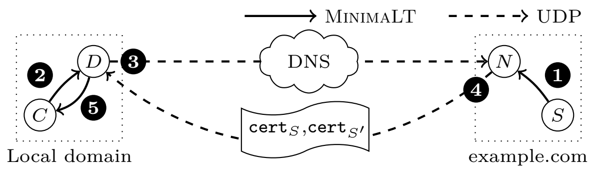


Figure 2: An external directory service query

sage to S; S receives this message and then completes the tunnel. (We describe this more completely in §5.) Tunnels provide server authentication, encryption, congestion control, and reliability; unlike with TLS/TCP, these services do not repeat for each individual connection.

3.3 Connections

A MinimaLT tunnel contains a set of **connections**, that is, a single tunnel between two hosts encapsulates an arbitrary number of connections. Each connection is user-authenticated and provides two-way communication between a client application and a service. In addition to multiplexing any number of standard application-to-service connections, each MinimaLT tunnel has a single **control connection**, along which administrative requests flow (§4.1).

3.4 Directory and Name service

Central to MinimaLT are its directory and name services. These services provide certificates called **service records**, which contain the information that a client needs to authenticate and communicate with a server. A **directory service** receives client queries containing a server hostname, and responds with the server's service record. Servers register their own service record using a **name service**. Thus the hosts involved in the authentication of a server are the client C, server S, directory service D, and name service N.

Contents of a service record A service record is made up of two certificates: a long-term certificate `cert`$_S$ and an ephemeral certificate `ecert`$_S$. The long-term certificate binds S's long-term public key with S's hostname. The ephemeral certificate is signed by S's long-term key and contains S's IP address, UDP port, protocol version number (now 0), minimum first packet size (padding), long-term key, ephemeral key, and the ephemeral key's lifetime. Thus this certificate includes the information which is traditionally returned by DNS, plus cryptographic parameters which in other protocols are negotiated during a handshake. The protocol version number allows algorithm upgrades; protocol changes always involve changing the version number.

Hosts Within an organization, an administrator maintains clients and servers, as well as one directory and one name service. Each server S periodically uploads S's service record to its local N. Before communicating with S, the client C requests S's service record from its local D. Once a client has S's service record, it can form a tunnel to S.

We note that *only* the interactions C–S, C–D, and S–N are part of MinimaLT. The communication D–N is not part of the MinimaLT protocol. We describe, for completeness, an implementation that uses DNS. The implementation is very simple: D does a DNS lookup on the hostname which returns S's service record. No changes are needed to DNS software: MinimaLT only adds service records as DNS TXT records.

Establishing a connection We depict an external lookup in Figure 2. The full sequence is as follows:

(1) S sends its service record to its local N;
(2) C requests S's service record from its local D (if D has cached S's service record, then skip to step 5);
(3) D makes a DNS query for S to N;
(4) N replies to D with S's service record; and
(5) D replies to C with S's service record.

Finally, after verifying the signatures on S's service record, C establishes a tunnel with S.

In successive sections, we will ignore steps (3) and (4) which are not part of the MinimaLT protocol. Further information on communication between C and D is given in §5.2 and between S and N is given in §5.4.

Discussion The directory service is designed for easy deployability on the Internet today while guaranteeing at least as much security as is currently obtained from the X.509 PKI used in TLS. In particular, a client C checks an X.509 certificate chain leading to the long-term public key for server S, in the same way that web browsers today check such chains. This chain is transmitted over DNS, obtaining three benefits compared to transmitting the chain later in the protocol: (1) The chain automatically takes advantage of DNS caching. (2) Even when not cached, the latency of transmitting the chain is usually overlapped with existing latency for DNS queries. (3) Any security added to DNS automatically creates an extra obstacle for the attacker, forcing the attacker to break *both* DNS security and X.509 security.

For comparison, if a client obtains merely an IP address from DNS and then requests an X.509 certificate chain from that IP address (the normal use of TLS today), then the attacker wins by breaking X.509 security alone. If a client instead obtains the S public key from DNS as a replacement for X.509 certificate chains then the attacker wins by breaking DNS security alone.

The integration of MinimaLT's directory services with DNS affects DNS configurations in two ways. First, the DNS record's time to live must be set to less than or equal to the key-erasure interval of the host it describes. We expect this to have a small impact, because most Internet traffic is to organizations that already use short times to live (e.g., 300 seconds for `www.yahoo.com` and `www.google.com`, and 60 seconds for `www.amazon.com`). Second, DNS replies will grow due to the presence of additional fields. The largest impact is the long-term certificate, which as mentioned above is encoded today as an X.509 certificate.

We do not claim that DNS and X.509 are satisfactory from a performance perspective or from a security perspective, but improved systems and replacement systems will integrate trivially with MinimaLT. We are planning a sayI implementation [53] which will have a stronger trust model and better performance.

4 MinimaLT packet format

Here we describe the MinimaLT packet format which we will build on in §5 to describe MinimaLT's packet flow. MinimaLT's simple packet format spans three protocol layers: the connection, tunnel, and delivery layers. We depict this format in Figure 3 and Table 1.

The plaintext portion of the packet contains the Ethernet, IP, and UDP headers; the Tunnel ID (TID); a number-used-once (nonce); and two optional fields that can be used during tunnel initiation—an ephemeral public key and a puzzle. A client provides the public key only on the first packet for

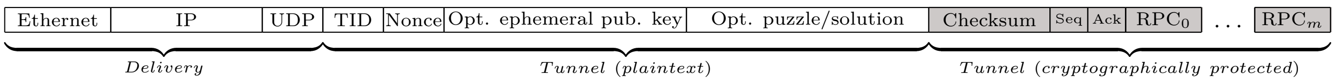


Figure 3: Packet format with plaintext in white and cryptographically protected portion in gray

	Field	Size (bytes) First	Size (bytes) Successive
Deliv.	Ethernet Header	14	14
	IP	20	20
	UDP	8	8
Rel. & Crypto.	Tunnel ID	8	8
	Nonce	8	8
	Ephemeral public key	32	n/a
	Puzzle/solution	148	n/a
	Checksum	16	16
	Sequence Num.	4	4
	Acknowledgment	4	4
Con.	Connection ID	4	4
	RPC	variable	
	Total (except RPC)	282	86

Table 1: Tunnel's first/successive packets

a tunnel, and a server requires puzzles opportunistically to prevent memory and computation attacks.

The packet's cryptographically protected portion contains ciphertext and the ciphertext's cryptographic checksum. The ciphertext contains a sequence number, acknowledgment number, and series of Remote Procedure Calls (RPCs).

4.1 RPCs

Each MinimaLT connection communicates a series of RPCs. An RPC is of the form $f_c(a_0, a_1, \ldots)$, where f is the name of the remote function, c is the connection on which the RPC is sent, and $a_0, a_1, \ldots$ are its arguments. On the wire this is encoded as $c, f, a_0, a_1, \ldots$ A single packet can contain multiple RPCs from one connection or from multiple connections within the same tunnel.

One connection is distinguished: connection 0 is the control connection, which hosts all management operations. These include creating, closing, accepting, and refusing connections; providing service records (§5.4); rekeying (§5.5) and IP address changes (§5.6); puzzles (§5.7); and flow control (§5.9). We reference the following RPCs:

RPC	Description
$\mathsf{create}_0(c, y)$	create anonymous connection c of type y
$\mathsf{createAuth}_0(c, y, U, x)$	create an authenticated connection for the user with long-term public key U, who generates authenticator x
$\mathsf{close}_0(c)$	close connection c
$\mathsf{ack}_0(c)$	creation of c successful
$\mathsf{refuse}_0(c)$	connection c refused
$\mathsf{requestCert}_0(S)$	get host S's service record
$\mathsf{giveCert}_0(\mathtt{cert}_S, \mathtt{ecert}_S)$	provide the service record for server S (contents described in §3.4)
$\mathsf{ok}_0()$	last request was OK
$\mathsf{nextTid}_0(t, C')$	advertise future TID to prepare for a rekey or IP address change (§5.5)
$\mathsf{rekeyNow}_0()$	server request for rekey
$\mathsf{puzz}_0(q, H(r), w, n')$	pose a puzzle (§5.7)
$\mathsf{puzzSoln}_0(r, n')$	provide a puzzle solution r
$\mathsf{windowSize}_0(c, n)$	adjust conn. receive window (§5.9)

All data on connections other than the control connection are sent unchanged to their corresponding applications.

In general, each service provided by a host supports a set of service-specific RPCs on standard connections. Our illustrations use the following sample RPC:

$\mathsf{serviceRequest}_c(\ldots)$	a request for some type of service on connection c

4.2 Authenticated encryption

MinimaLT is built on top of a high-level cryptographic abstraction, public-key authenticated encryption, to protect both confidentiality and integrity of messages sent from one public key to another. It is well known (see, e.g., [6, 7, 32, 25]) how to build public-key authenticated encryption on top of lower-level primitives: use static DH to assign a shared secret to the two public keys; use keys derived from the shared secret to encrypt each message and to authenticate each ciphertext. Note that low-level cryptographic message authentication should not be confused with user authentication and host authentication.

The input to public-key authenticated encryption is a plaintext to encrypt, the sender's secret key, the receiver's public key, and a nonce (a message number used only once for this pair of keys). The output is an authenticated ciphertext which is longer than the plaintext because it includes an authenticator (a cryptographic checksum). The inverse operation produces the original plaintext given an authenticated ciphertext, the same nonce, the receiver's secret key, and the sender's public key.

Each MinimaLT packet contains one authenticated ciphertext, together with the nonce used to create the ciphertext. The corresponding plaintext consists of reliability information (a sequence number and an acknowledgment number) and the concatenation of the RPCs sent in this packet.

MinimaLT chooses monotonically increasing nonces within each tunnel; our implementation uses time-based nonces. Once used, a nonce is never repeated for the same tunnel. The client uses odd nonces, and the server uses even nonces, so there is no risk of the two sides generating the same nonce. Clients enforce key uniqueness by randomly generating a new ephemeral public key for each new tunnel; this is a low-cost operation. For a host which operates as both client and server, its client ephemeral key is in addition to (and different from) its server ephemeral key.

MinimaLT uses the public-key authenticated encryption mechanism from NaCl [6], with 32-byte public keys, 16-byte authenticators, and 8-byte nonces. Except for these sizes and the performance reported in §6 our description of MinimaLT is orthogonal to this choice of encryption mechanism.

4.3 Tunnel IDs and ephemeral public keys

The tunnel establishment packet (the first packet sent between two hosts) contains a TID and the sending host's ephemeral public DH key. The TID is pseudo-randomly generated by the initiator. The public key is ephemeral to avoid identifying the client host to a third party.

Subsequent packets use the TID to identify the tunnel and thus do not repeat the DH key. The recipient uses the TID to look up the tunnel's shared secret used to verify and decrypt the authenticated ciphertext inside the packet. The TID is 64 bits—1/4 the size of a public key—with one bit indicating the presence of a public key in the packet, one bit indicating the presence of a puzzle/solution (see §5.7), and 62 bits identifying a tunnel.

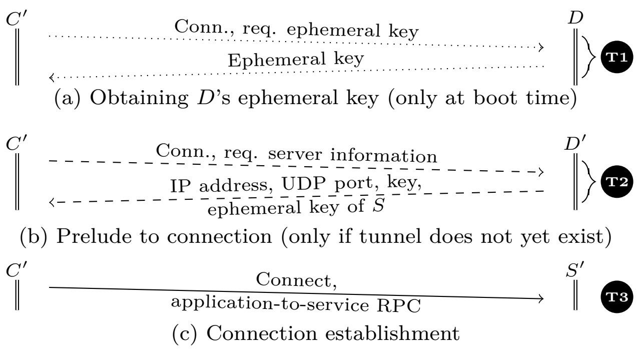

C′
Conn., req. ephemeral key
D
Ephemeral key
T1
(a) Obtaining D's ephemeral key (only at boot time)
C′
Conn., req. server information
D′
IP address, UDP port, key, ephemeral key of S
T2
(b) Prelude to connection (only if tunnel does not yet exist)
C′
Connect, application-to-service RPC
S′
T3
(c) Connection establishment

Figure 4: MinimaLT protocol trace

4.4 Delivery

MinimaLT concatenates tunnel information with the authenticated ciphertext and places the result into a UDP packet for delivery across existing network infrastructure. A MinimaLT packet on the wire thus contains standard Ethernet, IP, and UDP headers. The UDP header allows packets to traverse NATed networks [19] and enables user-space implementations of MinimaLT. Aside from the UDP length field, the delivery fields (including UDP port numbers) are ignored by MinimaLT. The MinimaLT protocol details are orthogonal to the structure of IPv4/v6 addresses.

5 MinimaLT packet flow

This section explains how a MinimaLT client C forms a tunnel and connects to a server S. All application data, as well as client and user identity, is encrypted under ephemeral keys that will soon be discarded. (IP addresses by necessity are visible.) Once a tunnel is formed, successive connections are created using RPCs within the existing tunnel.

We show a complete three-step connection in Figure 4: **T1** facilitates the request by C for D's ephemeral key (only required if this is C's first lookup since booting); **T2** protects the connection between C' and D' (used to request S's ephemeral key, the analogue of a DNS lookup); and **T3** protects communication between C' and S'. This section also covers more advanced issues such as rekeying and IP-address mobility; DoS protections; user authenticators; and congestion control.

We do not usually distinguish between a host (or user), their long-term public key, and their long-term private key, instead relying on context to disambiguate them. Figure 5 describes the notation we will use during the course of our description.

We show packets on a single line such as

$$t, n, C', \boxed{s, a, \ldots}\ {}^{C'\to S'}_{n}$$

which indicates a tunnel establishment packet (visible by the presence of the plaintext C', as described in §4.3) from C to S using keys C' and S' to box (encrypt and authenticate) the message '$s, a, \ldots$' under nonce n. Each packet has a new nonce but for conciseness we simply write n rather than n_1, n_2, etc. The same comment applies for sequence numbers (s) and acknowledgments (a).

5.1 Initial communication of C with D (T1)

After booting, C establishes a tunnel with D (Figure 4a, tunnel T1). (C's static configuration contains its directory service's IP address, UDP port, and long-term public key D.) C generates a single-use public key C' and uses it to create a bootstrap tunnel with the directory service.

$$t, n, C', \boxed{s, a, \mathsf{requestCert}_0(D)}\ {}^{C'\to D}_{n}$$

$$t, n, \boxed{s, a, \mathsf{giveCert}_0(\mathtt{cert}_D, \mathtt{ecert}_D)}\ {}^{D\to C'}_{n}$$

D responds with a service record ($\mathtt{cert}_D$,$\mathtt{ecert}_D$) containing its own ephemeral key, D'. C does not make further use of tunnel T1.

t	A tunnel ID (described in §4)
n	A nonce (described in §4)
s	A sequence number
a	An acknowledgment number
0 or c	A connection ID (0 for the control connection)
p	A puzzle
r	A puzzle solution (i.e., response)
C, D, N, S	The client, directory, name, and server long-term public/private key
C', D', N', S'	An ephemeral client, directory, name, and server public/private key
U	A user's public/private key
x	A user authenticator (described in §5.8)
$C \to S$	A message from the client to the server, using keys C and S
$H(m)$	The cryptographic hash of message m
$\boxed{m}\ {}^{k}_{n}$	Encrypt and authenticate message m using symmetric key k and nonce n
$\boxed{m}\ {}^{S\to P}_{n}$	Encrypt and authenticate message m using a symmetric key derived from private key S and public key P; n is a nonce

Figure 5: Notation

5.2 Subsequent communication of C with D (T2)

C uses D' to establish another tunnel with D (Figure 4b, tunnel T2) to request S's service record. This tunnel uses a fresh C', is key-erasure-protected, and is established by:

$$t, n, C', \boxed{s, a, \mathsf{requestCert}_0(S)}\ {}^{C'\to D'}_{n}$$

$$t, n, \boxed{s, a, \mathsf{giveCert}_0(\mathtt{cert}_S, \mathtt{ecert}_S)}\ {}^{D'\to C'}_{n}$$

Successive connections to the same server S skip both T1 and T2. T2 remains open for C to look up other servers.

5.3 Communication of C with S (T3)

After receiving $\mathtt{cert}_S$ and $\mathtt{ecert}_S$, C is ready to negotiate a tunnel with the end server (Figure 4, tunnel T3). C encrypts packets to the server using S' (from $\mathtt{ecert}_S$) and a fresh C'. Because C places its ephemeral public key in the first packet, both C and S can immediately generate a shared symmetric key using DH without compromising key erasure. Thus C can include application-to-service data in the first packet to S. That is,

$$t, n, C', \boxed{\begin{array}{l} s,\ a,\ \mathsf{nextTid}_0(t, C'), \\ \mathsf{createAuth}_0(1,\ \text{serviceName},\ U,\ x), \\ \mathsf{serviceRequest}_1(\ldots) \end{array}}\ {}^{C'\to S'}_{n}$$

Upon receiving $\mathsf{createAuth}_0$, S verifies the authenticator x (§5.8) and decides if the client user U is authorized to connect. If so, S creates the new connection, here with ID 1. The server ensures no two tunnels share the same C'. Once this tunnel is established, C erases the secret key belonging to C'; the tunnel continues to use the shared secret of C' and S'. (We describe the purpose of $\mathsf{nextTid}_0$ in §5.5.) The server can then immediately process the service-specific $\mathsf{serviceRequest}_1$ on this new connection.

5.4 Communication of S with N

Before a client may connect, S must register its own IP address, UDP port, public key S, certificate on S, and current ephemeral public key S' with an ephemeral-key name service

N. This is done using the $\mathsf{giveCert}_0$ RPC:

$$t, n, \boxed{s, a, \mathsf{giveCert}_0(\mathtt{cert}_S, \mathtt{ecert}_S)}\,{}^{S' \to N'}_{n}$$

$$t, n, \boxed{s, a, \mathsf{ok}_0()}\,{}^{N' \to S'}_{n}$$

(Here we assume S already has a tunnel to N.) In the local case N may be the same as D, but in general information must be pushed or pulled from N to D (§3.4).

5.5 Rekeying and key erasure

MinimaLT implements a property called **fast key erasure** which means that key erasure can take place at any time, without interrupting existing connections. A client uses a $\mathsf{nextTid}_0$ RPC to indicate the new TID t that will prompt a rekey; this RPC also includes a C' for reasons we describe in §7.4. Servers invoke $\mathsf{rekeyNow}_0$ when their key-erasure interval expires. In any case, each host erases the previous key after beginning to use the next. Clients invoke $\mathsf{nextTid}_0$ within each tunnel initiation packet (§5.3).

A client causes a rekey when its key-erasure interval expires or when it receives a $\mathsf{rekeyNow}_0$ RPC from the server. To cause such a rekey, the client (1) computes a new symmetric key by cryptographically hashing the tunnel's existing symmetric key; (2) computes a new $\overline{C'}$ indistinguishable from a legitimate public key; and (3) issues a tunnel initiation packet using t and $\overline{C'}$ which is encrypted with the new symmetric key and contains a successive $\mathsf{nextTid}_0$ RPC. The server (1) notices that the tunnel initiation packet contains the t from the previous $\mathsf{nextTid}_0$; (2) computes the new symmetric key and unboxes the packet; and (3) verifies that the current $\mathsf{nextTid}_0$'s $\overline{C'}$ parameter matches the plaintext public key on the tunnel initiation packet. If each verification succeeds, then the server transitions to the new tunnel parameters; otherwise it behaves as with a failed tunnel initialization.

5.6 IP-address mobility

Because MinimaLT identifies tunnels by their TID, a tunnel's IP and UDP port can change without affecting communication. After changing its IP address or UDP port, a client simply does a rekey.

5.7 Puzzles

A MinimaLT server under load that receives a tunnel establishment packet from a stranger client for an authorized service does not immediately create the tunnel. Instead, it produces a puzzle that must be solved by the client before the server will proceed. First, the server computes the solution (i.e., response) to a puzzle:

$$r = \boxed{C', S'}\,{}^{k}_{n'}$$

where k is a secret used only for puzzles. It then selects a value w that determines the difficulty of the puzzle, and calculates q by zeroing r's rightmost w bits. The server sends the puzzle of the form $[q, H(r), w, n']$ to the client in the following packet:

$$t, n, [q, H(r), w, n']$$

(Note that the square brackets do not affect the packet; they merely make our notation more readable.) The server next forgets about the client's request.

The client must solve the puzzle, i.e., reconstruct r from q and $H(r)$, and provide its solution r along with n' in a new tunnel establishment packet boxed using the same C' and S'. To find this solution, the client brute forces the rightmost w bits of q, checking each resulting candidate d to see if $H(d) = H(r)$. If this equivalence holds, the client has found $d = r$. (This brute force work imposes a high load on C without affecting S.) The client then responds to the server with:

$$t, n, C', [r, n'], \boxed{s, a, \ldots}\,{}^{C' \to S'}_{n}$$

To confirm a solution, the server decrypts r using k and n', confirms that the plaintext contains C' and S', and ensures that n' is within an acceptable window. Although the server had forgotten r, these protections ensure that r cannot be reused for other tunnel establishment attempts.

MinimaLT also provides a puzzle RPC that can be used inside a tunnel for liveness testing. The puzzle format used by this RPC is the same as that which is used in the tunnel header.

5.8 User authenticators

Every user serviced by MinimaLT is identified by his public key. The $\mathsf{createAuth}_0$ authenticator is the server's long-term public key encrypted and authenticated using the server's long-term public key, the user's long-term private key U, and a fresh nonce n never reused for U:

$$n, \boxed{S}\,{}^{U \to S}_{n}$$

The effect of user authenticators is determined by the server's local authorization policy.

5.9 Congestion and flow control

MinimaLT's tunnel headers contain the fields necessary to implement congestion control, namely sequence number and acknowledgment fields. We presently use a variation of TCP's standard algorithms [22].

MinimaLT hosts adjust per-connection flow control using the $\mathsf{windowSize}_0$ RPC. MinimaLT subjects individual connections to flow control, so $\mathsf{windowSize}_0$ takes as parameters both a connection ID and size. MinimaLT currently implements TCP-style flow control.

As with TCP [27], efficient congestion control is an area of open research [23], and we could substitute an emerging algorithm with better performance. MinimaLT does have one considerable effect on congestion control: controls are aggregated for all connections in a tunnel, rather than on individual connections. Since a single packet can contain data for several connections, the server no longer needs to allocate separate storage for tracking the reliability of each connection. This also means that MinimaLT need not repeat the discovery of the appropriate transmission rate for each new connection, and a host has more information (i.e., multiple connections) from which to derive an appropriate rate. The disadvantage is that a lost packet can affect all connections in aggregate.

6 Performance evaluation

In this section we evaluate MinimaLT's performance, specifically packet overhead, latency and throughput. For performance under DoS, see §7.6. We focus on server performance, because even resource-constrained smartphones can adequately handle their own load. While servers have faster CPUs, they are relatively slow compared to their network capacity and the number of clients they serve. Furthermore, a server DoS is much more damaging than a client DoS.

6.1 Packet header overhead

MinimaLT's network bandwidth overhead is modest. The overhead is due to the cryptography, and includes the nonce, TID, and checksum (the public key/puzzle fields are rarely present and are thus insignificant overall). MinimaLT requires 32 bytes more for its headers than TCP/IP; this represents 6% of the minimum Internet MTU of 576 bytes, and 2% of 1518-byte Ethernet packets.

6.2 Latency and throughput

We experimentally evaluate MinimaLT's performance in three areas: (1) the serial rate at which MinimaLT establishes tunnels/connections, primarily to study the effect of latency on the protocol; (2) the rate at which MinimaLT establishes tunnels/connections when servicing many clients in parallel; and (3) the throughput achieved by MinimaLT.

We benchmarked MinimaLT on Ethos. All of our performance tests were run on two identical computers with a 4.3 GHz AMD FX-4170 quad-core processor, 16GB of memory, and a Gb/s Ethernet adapter. We benchmarked in 64-bit mode and on only one core to simplify cross-platform comparisons.

Serial tunnel/connection establishment latency In each of our serial connection benchmarks, a client sequentially connects to a server, sends a 28-byte application-layer request, and receives a 58-byte response. We measure the number of such operations completed in 30 seconds; the connections do not perform a DNS/directory service lookup. We performed this experiment under a variety of network latencies using Linux's netem interface.

We compare against OpenSSL 1.0.0j using its s_server and s_time utilities, running on version 3.3.4 of the Linux kernel. We configured OpenSSL to use 2,048-bit RSA as recommended by NIST [4] (although 2,048-bit RSA provides 112-bit security, less than that of MinimaLT's 128-bit security), along with 128-bit AES, ephemeral DH, and client-side certificates (i.e., cipher suite DHE-RSA-AES128-SHA). In order to ensure disk performance did not skew our results, we modified s_server to provide responses from memory instead of from the filesystem. We also wrote a plaintext benchmark which behaves similarly, but makes direct use of the POSIX socket API without cryptographic protections.

To produce results analogous to OpenSSL, we simulated both (1) many abbreviated connection requests for high reuse by clients and (2) many full connection requests for no reuse by clients, we tested both (1) the vanilla MinimaLT stack and (2) a MinimaLT stack we modified to artificially avoid tunnel reuse.

Figure 6a shows full connection performance ratios between MinimaLT vs. TCP and MinimaLT vs. OpenSSL. For each connection, MinimaLT creates a new tunnel and authenticates the client user; TCP performs a three-way handshake; and OpenSSL performs full TCP and TLS handshakes. After this initial setup, our benchmark completes an application-data round trip. Most surprisingly, MinimaLT creates connections faster than unencrypted TCP, beginning before native LAN latencies+$1/2$ ms (LAN+$1/2$ ms). MinimaLT is about twice as fast as unencrypted TCP at latencies above LAN+4 ms, and MinimaLT is four to six times faster than OpenSSL.

Figure 6b shows abbreviated connections. Here MinimaLT reuses an existing tunnel and OpenSSL takes advantage of its session ID to execute an abbreviated connection. Both protocols avoid computing a shared secret using DH, except in the case of the first connection. Abbreviated MinimaLT connections surpass the performance of unencrypted TCP at LAN+$1/4$ ms latencies. At higher latencies, MinimaLT is two times the performance of TCP and three times the performance of OpenSSL.

At high latencies, the protocol performance ratios approach the numbers predicted by counting the round trips inherent in each protocol. We attribute our results to MinimaLT's efficient tunnel/connection establishment (especially at high latencies) and to the speed of the NaCl library (especially at low latencies).

Figure 6c revisits the measurements described for Figure 6a, this time plotting the actual time each connection took in log scale. At LAN+$1/16$ ms, a MinimaLT, TCP, and OpenSSL connection took 1.32ms, 0.68ms, and 7.63ms, respectively. At LAN+256ms, the times were 0.53s, 1s, and 2.13s, respectively.

Figure 6d displays abbreviated connection time. At LAN+$1/16$ ms, a MinimaLT and OpenSSL connection took 1.03ms and 1.67ms, respectively. At LAN+256ms, the times were 0.52s and 1.60s, respectively. (TCP times are the same as in Figure 6c.)

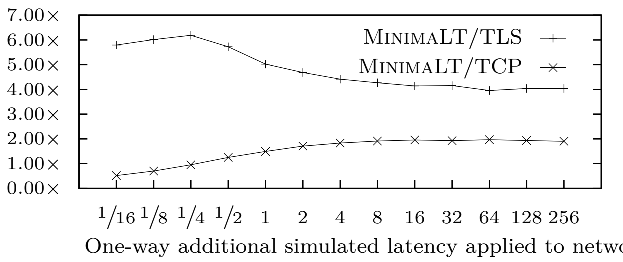
Tunnels per run	**User Auth.**	**Connections per second**	**DH per conn.**
One		18,453	0
One	✓	8,576	1
Many		7,827	1
Many	✓	4,967	2

Table 2: Connection establishment with many clients

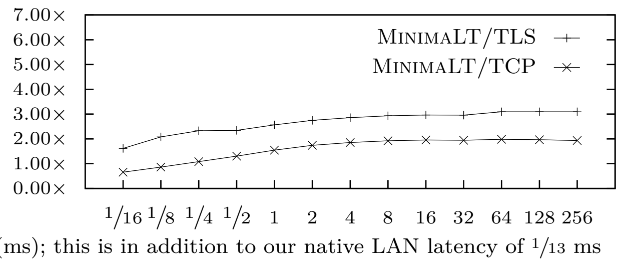
Tunnel establishment throughput with many clients We created a second connection benchmark to estimate the CPU load on a MinimaLT server servicing many clients. To do this, we ran two client OS instances, each forking several processes that connected to the server repeatedly as new clients; here each virtual machine instance was running on a distinct processor core within a single computer. Because this experiment concerns CPU use and not latency, these clients do not write any application-layer data, they only connect. Using xenoprof and xentop, we determined that the server was crypto-bound (i.e., 63% of CPU use was in cryptography-related functions—primarily public key—and the server CPU load was nearly 100%). We measured the number of full connections per second achieved under this load, and varied our configuration from accepting fully-anonymous users (no authenticators), to verifying a new user authenticator for each connection request. Our MinimaLT server established 4,967–7,827 tunnels per second, as shown in Table 2 (rows 3–4).

Given the minimal tunnel request size of 1,024 bytes, our hosts can (on a single core) service 61.15Mb/s of tunnel requests from anonymous users and at least 38.80Mb/s of tunnel requests from authenticated users. We note that the latter is the worst case for authenticated users; in general, we would cache the result of the DH computations necessary to validate user authenticators, as authenticators use long-term keys. Thus in practice we expect the authenticated user case to approach that of anonymous users.

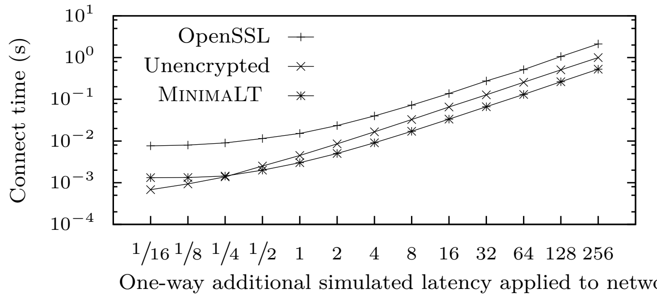
Connection establishment throughput with many clients We repeated the previous experiment, but this time avoided DH computations by repeatedly creating connections using a single tunnel between each client and the server. Table 2 (rows 1–2) shows that our rates ranged from 8,576–18,453 per second, depending on the presence of user authenticators. The connection rate over a single tunnel is important for applications which require many connections



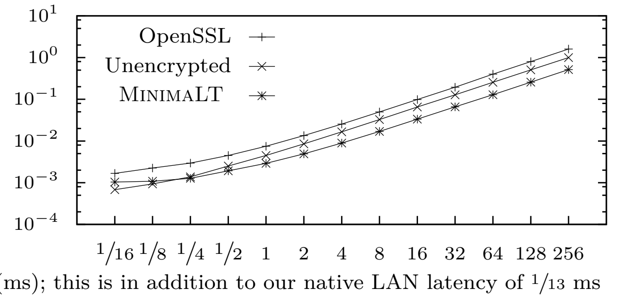
(a) MinimaLT gain over TCP/IP and TLS; full connections



(b) MinimaLT gain over TCP/IP and TLS; abbr. connections



(c) Time spent creating a connection; full connections



(d) Time spent creating a connection; abbr. connections

Figure 6: Serial tunnel/connection establishment latency

and when using many applications to communicate with the same server. Our comments about cached DH results in the preceding experiment apply here as well; we would expect in practice the rate of the authenticated case will approach the anonymous case.

A theoretical throughput limit We used SUPERCOP [5] to measure the time it takes our hardware to compute a shared secret using DH, approximately 293,000 cycles or 14,000 shared-secret computations per second. MinimaLT's tunnel establishment rate approaches 56% of this limit, with the remaining time including symmetric key cryptography, scheduling, and network stack processing.

System	Bytes per second
Line speed	125,000,000
Unencrypted	117,817,528
MinimaLT	113,945,258
OpenSSL	111,448,656

Table 3: Data throughput (ignoring protocol overhead)

Single-connection data throughput Table 3 describes our throughput results, observed when running programs that continuously transmitted data on a single connection for thirty seconds. MinimaLT approaches the throughput achieved by unencrypted networking and runs at 91% of line speed (Gb/s). Indeed, MinimaLT's cryptography runs at line speed; header size differences are the primary reason the unencrypted benchmark outperforms MinimaLT.

7 Design rationale

7.1 Cryptographic security

Tunnel IDs and nonces are visible on the network, and follow a clear pattern for each client-server pair. Essentially the same information is available through a simple log of IP addresses of packets sent. Mobile clients automatically switch to new tunnel IDs when they change IP addresses, as we described in §5.6. Ephemeral client public keys are visible when each tunnel is established, but are not reused and are not connected to any other client information.

Other information is boxed (encrypted and authenticated) between public keys at the ends of the tunnel. These boxes can be created and understood using either of the two corresponding private keys, but such keys are maintained locally inside MinimaLT hosts. The attacker can try to violate confidentiality by breaking the encryption, or violate integrity by breaking the authentication, but MinimaLT uses modern cryptographic primitives that are designed to make these attacks very difficult. The attacker can also try to substitute his public key for a legitimate public key, fooling the client or server into encrypting data to the attacker or accepting data actually from the attacker, but this requires violating integrity of previous packets: for example, before the client encrypts data to D', the client obtains D' from a boxed packet between D and C'. MinimaLT's reliability and connection headers are part of the ciphertext, so they are also protected against tampering and eavesdropping.

7.2 The benefits of RPCs

In contrast to byte-oriented protocols, we use RPCs within MinimaLT's connection layer as they result in a clean design and implementation; they are also general and fast (§6). RPCs have a long history dating to the mid-1970s [60, 9].

The tunnel, and the RPCs within it, are totally sequenced; thus RPCs are executed in order (as opposed to separately implemented connections—as in TLS—where ordering between connections is not fixed). This enables a clean separation of the control connection from other connections, and we have found that this simplifies both the protocol and its implementation. Furthermore, placing multiple RPCs in one packet amortizes the overhead due to MinimaLT's delivery and tunnel fields across multiple connections.

7.3 The benefits of tunnels

Tunnels make it more difficult for an attacker to use traffic analysis to infer information about communicating parties. Of course, traffic analysis countermeasures have limits [43]; for obvious cost reasons we did not include extreme protections against traffic analysis, such as using white noise to maintain a constant transmission rate. MinimaLT also reduces packet overhead by using TIDs rather than repeating the public key in every packet.

Tunnels are independent of the IP address of C; this means that C can resume a tunnel after moving to a new location (typically prompting the use of the next ephemeral key as described below), avoiding tunnel-establishment latency and application-level recovery due to a failed connection. This reduced latency is useful for mobile applications, in which IP-address assignments may be short-lived, and thus overhead may prevent any useful work from being done before an address is lost.

MinimaLT reduces an attacker's ability to link tunnels across IP address changes because its TID changes when its IP address changes. What remains is temporal analysis, where an attacker notices that communication on one IP address stops at the same time that communication on another starts. However, the attacker cannot differentiate for sure between IP mobility and an unrelated tunnel establishment. Blinding information below the network layer—for example, the Ethernet MAC—is left to other techniques.

7.4 Rekeying

Rekeying is critical for allowing key erasure during a long connection: it allows clients and servers to periodically forget old encryption keys, protecting those keys against an attacker who later compromises clients and servers. Rekeying also supports IP-address mobility without explicitly linking the old address to the new address.

Creating the new key by hashing is more efficient than creating it by a new DH, avoiding both public-key operations and superfluous round trips.

The client creates a new public key, so that there is no packet-format distinction between a rekeying tunnel-initiation packet and a completely new tunnel-initiation packet. Rather than generating a key pair and using only the public key, a public key can directly and more efficiently be generated as a random point on an elliptic curve.

There are two copies of C' in a rekeying packet, one in the clear and one inside the boxed part, verified by the server to be identical. Without this verification, an active attacker could modify the public key sent in the clear, observe that the server still accepts this packet, and confidently conclude that this is a rekey rather than a completely new tunnel.

A client-side administrator sets his host's key-erasure interval as a matter of policy. The server's policy is slightly more sensitive, because the server must maintain its ephemeral key pairs as long as they are advertised through the directory service. An attacker who seizes a server could combine the ephemeral keys with captured packets to regenerate any symmetric key within the ephemeral key window. Thus even if the client causes a rekey, the server's ephemeral key window dominates on the server side.

If a user believes a server's ephemeral key lifetime is long enough to put his communications at risk, then he could choose not to communicate with the server. Except for the case of a malicious server which does not destroy expired ephemeral keys (and ought not to be trusted in the first place), MinimaLT's directory service provides the lifetime information needed to make these decisions, whether directly by a human or by automated tools.

7.5 User authenticators

Because authenticators are transmitted inside boxes (as ciphertext), they are protected from eavesdropping, and because the authenticator is tied to the server's (certified) public key, the server cannot use it to masquerade as the user to a third-party MinimaLT host. Of course, any server could choose to ignore the authenticator or perform operations the client did not request, but that is unavoidable. If third-party auditability is desired then users can choose to interact only with servers that take requests in the form of certificates.

7.6 Denial of service

DoS protections in MinimaLT are intended to maintain availability against much more severe attacks than are handled by current Internet protocols. Of course, an exceptionally powerful attacker will be able to overwhelm a MinimaLT server, but DoS protections are useful even in such extreme situations as a way to consume the attacker's resources and limit the number of DoS victims. Of particular concern are DoS attacks which consume memory or computational resources; the protocol cannot directly defend against network exhaustion attacks, although it avoids contributing to such attacks by preventing amplification.

We introduced anonymous and stranger-authorized services in §3.1. Anonymous services (i.e., permit create_0) perform a DH computation to compute a shared secret and maintain tunnel data structures that consume just under 5KB each (this is configurable; most memory use is due to incoming and outgoing packet buffers). Stranger-authorized services (i.e, require $\mathsf{createAuth}_0$, but permit strangers) additionally perform a public-key decryption to validate each new user authenticator encountered. MinimaLT puzzles serve as a countermeasure to DoS attacks on these services [33], and we now describe how they are applied at key points of the MinimaLT protocol. (Recall that administrators can additionally address DoS attacks from *known* users through de-authorization or non-technical means.)

Before establishing a tunnel In the case of anonymous services, a single attacker could generate a large number of ephemeral public keys to create many tunnels, with each tunnel consuming the resources described above. Furthermore, the attacker's host might avoid creating a tunnel data structure or performing any cryptographic operations, thus making the attack affect the server disproportionately.

MinimaLT addresses these attacks using puzzles present in its tunnel headers. Servicing tunnel requests in excess of the limits discussed in §6.2 would cause a server to require these puzzles, and because they take $O(2^{w-1})$ operations for a client to solve, the server can require clients to pay a wide range of computational costs to connect, here w is a 32-bit value. On the other hand, puzzles are of little burden on the server; our test hardware can generate and verify 386,935 puzzles per second. Since a puzzle interrogation and padded solution packet are 206 and 1,024 bytes, respectively, a single CPU core can verify puzzles at 394% of Gb/s line speed.

Clients overloaded by puzzles (possibly forged by a MitM attacker) can choose how to allocate CPU time, for example prioritizing connections that succeed without puzzles.

Amplification attacks against third parties At tunnel establishment, an anonymous or stranger-authorized MinimaLT service might respond to packets from clients which spoof another host's IP address. This is always possible with the directory service, which initially must react to a request from an unknown party before transitioning to key-erasure-protected authentication. A MitM could spoof the source of packets and even complete a successive puzzle interrogation. A weaker attacker could elicit a response to the first packet sent to a server. Given this, MinimaLT is designed to minimize amplification attacks, in which a request is smaller than its reply (to a spoofed source address). A connection request causes a connection acknowledgment or puzzle interrogation; both responses are smaller than the request. Flow-control acknowledgments are randomized so that blind clients cannot request further packets.

After establishing a tunnel Given a tunnel, an attacker can easily forge a packet with garbage in place of ciphertext and send it to a service. This forces MinimaLT to decrypt the packet and verify its checksum, wasting processor time. However, MinimaLT's symmetric cryptography on established tunnels operates at line speed on commodity hardware (§6.2), so DoS would be equivalent to the attacker exhausting the network.

MinimaLT can send puzzle RPCs arbitrarily, so a server can use low cost (i.e., small w) puzzles to check whether clients remain available, and then garbage collect idle connections. Additionally, a server can increase the value of w to make clients pay a computational cost to keep a connection alive. We use cryptographically protected RPCs to pose and solve these puzzles to prevent an attacker from attempting RST-style mischief [18].

Creation of fictitious strangers Stranger services are vulnerable to further CPU attacks—attackers could generate false user identities that would fail authentication, but only after the server performed a public-key decryption. A server will apply the puzzle RPCs when connection rates exceed the limits discussed in §6.2.

An attacker could also generate verifiable authenticators and connect to a stranger-authorized service many times as different stranger users. This would cause a system to generate accounts for each stranger identity. However, this is no different from any other creation of pseudo-anonymous accounts; it is up to the system to decide how to allocate account resources to strangers. Perhaps the rate is faster, but unlike many contemporary pseudo-anonymous services, a MinimaLT system can prune stranger accounts as necessary; the stranger's long-term resources (e.g., files on disk) will remain isolated and become available if the account is later regenerated because public keys remain temporally unique [50]. Of course, applications could impose additional requirements (e.g., a Captcha) before allowing a stranger to consume persistent resources like disk space.

8 Comparison to previous work

Table 4 compares MinimaLT to several earlier Internet protocols. MinimaLT is unique in that it provides encryption and authentication with fast key erasure while allowing a client to include data in the first packet sent to a server (often forgoing pre-transmission round trips entirely). MinimaLT also contains robust DoS protections.

We have omitted one very recent protocol from Table 4: Google's QUIC, which was developed independently of and concurrently with MinimaLT. Our preliminary assessment of the QUIC protocol documentation (released in late June 2013) is that QUIC uses some of the same latency-reducing techniques as MinimaLT, but does not overlap DNS/directory service lookups with tunnel establishment. QUIC also appears to have lower security goals than MinimaLT: for example, allowing earlier data in long-term connections to be retroactively decrypted.

8.1 Security advantages over TLS

TLS [15] is widely deployed as the primary mechanism for securing Internet communication across administrative domains, and in particular as the primary network security layer in web browsers. The importance of TLS warrants a multifaceted comparison of TLS with MinimaLT. We already showed in §6 that MinimaLT is more efficient than TLS, but we also claim that MinimaLT has several important security advantages over TLS, as discussed below.

8.1.1 Cryptographic abstractions TLS builds on separate cryptographic primitives for public-key cryptography, secret-key encryption, etc. Unfortunately, composing these low-level primitives turns out to be complicated and error-prone. For example, the BEAST attack [17] and the very recent Lucky 13 attack [2] recovered TLS-encrypted cookies by exploiting the fragility of the "authenticate-pad-encrypt" mechanism used by TLS to combine secret-key encryption with secret-key authentication. TLS implementations have worked around these particular attacks by (1) sending extra packets to hide the "IV" used by BEAST and (2) modifying implementations to hide the timing leaks used by Lucky 13; however, further attacks would be unsurprising.

The modern trend is for cryptographers to take responsibility for providing secure higher-level primitives. For example, cryptographers have defined robust high-performance "AEAD" primitives that handle authentication and encryption all at once using a shared secret key [45], taking care of many important details such as padding and key derivation. This simplifies protocol design, eliminating the error-prone step of having each protocol combine separate mechanisms for authentication and encryption. TLS 1.2 (not yet widely deployed) supports AEAD primitives.

MinimaLT is built on top of an even higher-level primitive, *public-key* authenticated encryption, as mentioned in §4.2. This further simplifies protocol design.

8.1.2 Verifiability One might think that existing protocol-analysis tools are already powerful enough to formally verify the confidentiality and integrity properties of a clean high-level protocol such as MinimaLT, assuming that the underlying cryptographic primitives are secure. However, the security properties of authenticated encryption using non-interactive DH were only very recently formalized (see [25]), and more work is required to develop a higher-level security calculus on top of these properties; note that replacing authenticated encryption with *unauthenticated* encryption would eliminate the security of typical protocols that use authenticated encryption. Thus, we do not claim that MinimaLT is formally verified.

However, we do claim that MinimaLT will be far easier to verify than TLS, and that there are far fewer opportunities for mistakes in MinimaLT than in TLS. Attempts to verify the security of TLS (such as [32]) have so far covered only limited portions of TLS, and have not prevented a seemingly never-ending string of announcements of TLS security failures, such as the BEAST and Lucky 13 attacks cited above. The unverified portions of TLS are more complex than the entire MinimaLT protocol.

8.1.3 Security goals TLS is normally implemented as a user-space library that adds cryptographic network protections to an insecure transport layer, TCP. This structure prevents TLS from providing strong protection against DoS: packets that deny service at the TCP layer are not even seen by TLS.

TLS can provide some key erasure through the use of ephemeral DH. Here the server generates a new DH key pair for each TLS session, and it uses this key pair to negotiate the session key with the client [38]. This means that past session keys remain secret even if a server's long-term key is compromised. On the other hand, a TLS session itself might remain in use for a long time, and session keys are obviously vulnerable to physical compromise as long as they are in use. This effect is compounded when servers support abbreviated connections [39]. In contrast, MinimaLT implements its key erasure using periodic rekeys, as we described in §5.5. Placing a time limit on keys simplifies security analysis because it removes the effect of variable session lifetimes.

8.1.4 Robustness Many Internet applications avoid the use of TLS or use weak TLS options [58]. Even well-meaning developers routinely misuse complex TLS APIs, resulting in security holes [26, 21]. Optionally, TLS can provide user-level authentication using client-side certificates, but authorization is left to application logic. MinimaLT forgoes backwards compatibility to provide a simpler, less mistake-prone platform, and it subsumes much of the work traditionally left to application programmers.

Another benefit of MinimaLT's clean-slate design is a simpler code base. OpenSSL contains 252,000 C Lines of Code (LoC). Much of this code might not be used, depending on how a service is configured or because it also implements utilities and benchmarks, but our study found 74,000 LoC associated with the DHE-RSA-AES128-SHA cipher suite. Additionally, there is code in the OS to support TCP/IP. In contrast, MinimaLT's design results in a protocol code base of 12,000 LoC along with NaCl, where our choice of ciphers uses another 6,920 LoC.

8.2 Advantages over other protocols

TCP's three-way handshake establishes a random Initial Sequence Number (ISN). This is necessary for two reasons: (1) the ISN serves as a weak authenticator (and liveness check) because a non-MitM attacker must predict it to produce counterfeit packets, and (2) the ISN reduces the likelihood that a late packet will be delivered to the wrong application.

MinimaLT encrypts the sequence number, provides cryptographic authentication, and checks liveness using puzzles, addressing (1). MinimaLT uses TIDs, connection IDs, and nonces to detect late packets, addressing (2). Thus MinimaLT can include application data in a connection's first packet, as discussed above, eliminating the need for a transport-layer three-way handshake. Extra round trips are necessary only if the tunnel does not exist; and then only when the client does not have S's service record or is presented with a puzzle. If the server provides a puzzle, it means that the server is under heavy load so that additional latency is unavoidable.

TCP Fast Open (TFO) [48] clients can request a TFO cookie that allows them to forgo TCP's three-way handshake on future connections. However, since any client may request a TFO cookie, a client may spoof its sending IP address to mount a DoS attack against a server; under this condition, the server must again require a three-way handshake. To benefit from TFO, a server application must be idempotent, a requirement that MinimaLT avoids.

Structured Stream Transport (SST) [24] allows applications to associate lightweight network streams with an existing stream, reducing the number of three-way handshakes incurred by applications and providing semantics useful for applications that use both data and control connections. MinimaLT eliminates the handshake on even the first connection, and MinimaLT's tunnels do not require a programmer's explicit use of a lightweight stream API.

Internet Protocol Security (IPsec) provides very broad confidentiality and integrity protections because it is generally implemented in the OS kernel. For example, IPsec can be configured such that *all* communication between node A and node B is protected. This universality simplifies assurance. IPsec also provides fast key erasure at the expense of a DH computation [34]. Many key management protocols have been proposed for IPsec; we were particularly inspired by Just Fast Keying due to its simplicity, focus on forward secrecy, and DoS resilience [1]. IPsec's major shortcoming is that its protections stop at the host; it focuses on network isolation and host authentication/authorization. For example, IPsec does not authenticate or authorize users.

Labeled IPsec [31] combines IPsec and Security-Enhanced Linux (SELinux) [44] to provide more comprehensive network protections. Using a domain-wide authorization policy, the system (1) associates SELinux labels with IPsec security associations, (2) limits a process's security associations (connections) using a kernel authorization policy, and (3) employs a modified inetd that executes worker processes in a security domain corresponding to the label associated with an incoming request. In this manner, labeled IPsec can solve many of the authentication deficiencies in plain IPsec. However, labeled IPsec builds upon the Linux kernel, SELinux, and IPsec, each of which are very complex. Furthermore, IPsec security association granularity limits the granularity of controls in labeled IPsec. In contrast, MinimaLT is designed from scratch, significantly simplifying policy specification, implementation, and use.

Many researchers have attempted to reduce the latency inherent in TLS and TCP. False Start (no longer used), Snap Start, and certificate pre-fetching have accelerated establishing a TLS session [41, 37, 56]. Datagram Transport Layer Security (DTLS) [49] provides TLS protections on top of UDP, which is useful when reliability is unnecessary. However, DTLS shares TLS' initial handshake latency.

Like MinimaLT, tcpcrypt [10] investigated ubiquitous encryption, but it maintains backwards compatibility with TCP. Tcpcrypt provides hooks that applications may use to provide authentication services and determine whether a channel is encrypted. MinimaLT's approach is different; it is clean-slate and eases host assurance by moving authentication and encryption services to the system layer.

Stream Control Transmission Protocol (SCTP) provides reliable delivery and congestion control [57], but it differs from TCP in that it can bundle messages from multiple applications (i.e., *chunks*) into a single packet. MinimaLT borrows this technique.

9 Conclusion

MinimaLT provides network confidentiality, integrity, privacy, server authentication, user authentication, and DoS protections with a simple protocol and implementation.

	TCP	TCP Fast Open	SST	IPsec	Labeled IPsec	TLS	False Start	Snap Start	Tcpcrypt	MinimaLT
Encrypt			✓	✓	✓	✓	✓	✓	✓	✓
Key erasure after session				✓	✓	✓	✓	✓	✓	✓
User authentication					✓	✓	✓	✓	✓	✓
Fast (time-based) key erasure				✓	✓					✓
Robust DoS protections										✓
Round trips before client sends data*	2	2	2	$\geq$4	$\geq$4	4	3	2	3	1
. . . if server is already known	1	1	1	$\geq$3	$\geq$3	3	2	1	2	0
. . . in abbreviated case†	1	0	0	1	1	2	2	1	1	0

*Includes one round trip for DNS/directory service lookup of unknown server
†Assumes protocol-specific information cached from previous connection to same server

Table 4: Comparison of MinimaLT with other network protocols

A particular concern for protected networking is latency, as research has shown users are very sensitive to delay. MinimaLT combines directory services and tunnel establishment in a new way to minimize latency—even outperforming unencrypted TCP. MinimaLT's first round trip is performed only once, at system boot time. The second is a protected analogue of a DNS lookup and is required under the same circumstances as DNS. Thus in the typical case, MinimaLT clients transmit encrypted data to an end server in the first packet sent.

MinimaLT establishes a tunnel which can be long-lived. Of course, the tunnel can be terminated at any time, but absent resource constraints MinimaLT is intended to maintain tunnels even across system suspends and network migration. This makes for a more reliable system as recovery code needs to be run less often.

Future work will focus on increasing tunnel establishment rates by offloading public key operations to other CPU cores. We expect a roughly N-fold improvement in cryptography from using N cores, and thus expect Gb/s-speed tunnel establishment with 16 cores. (When not under attack, MinimaLT would use far fewer cores.) We also plan to build proxies which will enable MinimaLT to talk to legacy applications. We expect to soon release Ethos and our Linux MinimaLT implementation as open source software. See `http://www.ethos-os.org/software.html`.

10 Acknowledgments

This material is based upon work supported by the US National Science Foundation under grants CNS-0964575 and 1018836, the European Commission under Contract ICT-2007-216676 ECRYPT II and INFSO-ICT-284833 PUFFIN, and by the Netherlands Organisation for Scientific Research (NWO) under grant 639.073.005. Volker Roth, our shepherd, provided extensive feedback, especially to improve the paper's organization. We also thank Wenyuan Fei, Yaohua Li, and Siming Chen for their reviews of initial drafts, our anonymous referees for comments on subsequent drafts, and the United States Military Academy and Army Cyber Center for their support.

11 References

[1] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. D. Keromytis, and O. Reingold. Just Fast Keying: Key agreement in a hostile Internet. *ACM Trans. Inf. Syst. Secur.*, 7(2):242–273, May 2004.

[2] N. AlFardan and K. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, pages 526–540, Washington, DC, USA, May 2013. IEEE Computer Society Press.

[3] K. J. Argyraki, P. Maniatis, O. Irzak, S. Ashish, and S. Shenker. Loss and delay accountability for the internet. In *Proceedings of the 2007 International Conference on Network Protocols*, pages 194–205, Washington, DC, USA, 2007. IEEE Computer Society Press.

[4] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for key management—Part 1: General (revised). US National Institute of Standards and Technology, Mar. 2007. `http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf` (accessed Aug 26, 2013).

[5] D. J. Bernstein and T. Lange. eBACS: ECRYPT Benchmarking of Cryptographic Systems. `http://bench.cr.yp.to/` (accessed Aug 26, 2013).

[6] D. J. Bernstein, T. Lange, and P. Schwabe. NaCl: Networking and cryptography library. `http://nacl.cr.yp.to/` (accessed Aug 26, 2013).

[7] D. J. Bernstein, T. Lange, and P. Schwabe. The security impact of a new cryptographic library. In *International Conference on Cryptology and Information Security in Latin America*, volume 7533, pages 159–176. Springer, 2012.

[8] D. J. Bernstein and P. Schwabe. NEON crypto. In *Workshop on Cryptographic Hardware and Embedded Systems*, volume 7428, pages 320–339. Springer, 2012.

[9] A. Birrell and B. J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, Feb. 1984.

[10] A. Bittau, M. Hamburg, M. Handley, D. Mazières, and D. Boneh. The case for ubiquitous transport-level encryption. In *Proceedings of the 19th USENIX Security Symposium*, Berkeley, CA, USA, Aug. 2010. USENIX Association.

[11] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, pages 553–567, Washington, DC, USA, May 2012. IEEE Computer Society Press.

[12] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The information visualizer, an information workspace. In *Proceedings of the 1991 Conference on Human Factors in Computing Systems*, pages 181–188, New York, NY, USA, Apr. 1991. ACM.

[13] L. Constantin. Facebook to roll out HTTPS by default to all users, Nov. 2012. `http://www.computerworld.com/s/article/9233897/Facebook_to_roll_out_HTTPS_by_default_to_all_users` (accessed Aug 26, 2013).

[14] M. de Vivo, G. O. de Vivo, R. Koeneke, and G. Isern. Internet vulnerabilities related to TCP/IP and T/TCP. *SIGCOMM Comput. Commun. Rev.*, 29(1):81–85, Jan. 1999.

[15] T. Dierks and C. Allen. RFC 2246: The TLS protocol version 1, Jan. 1999. Status: PROPOSED STANDARD.

[16] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, Berkeley, CA, USA, Aug. 2004. USENIX Association.

[17] T. Duong and J. Rizzo. Here come the ⊕ ninjas. In *Ekoparty Security Conference*, 2011.

[18] P. Eckersley, F. von Lohmann, and S. Schoen. Packet forgery by ISPs: A report on the Comcast affair. Electronic Frontier

Foundation, Nov. 2007.
https://www.eff.org/files/eff_comcast_report.pdf (accessed Aug 26, 2013).

[19] K. Egevang and P. Francis. RFC 1631: The IP network address translator (NAT), May 1994. Status: INFORMATIONAL.

[20] Electronic Frontier Foundation. HTTPS everywhere. https://www.eff.org/https-everywhere (accessed Aug 26, 2013).

[21] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben. Why Eve and Mallory love Android: an analysis of Android SSL (in)security. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, pages 50–61, New York, NY, USA, 2012. ACM.

[22] S. Floyd. RFC 2914: Congestion control principles, Sept. 2000. Status: INFORMATIONAL.

[23] B. Ford. Directions in Internet transport evolution. *IETF Journal*, 3(3):29–32, Dec. 2007.

[24] B. Ford. Structured streams: a new transport abstraction. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 361–372, New York, NY, USA, 2007. ACM.

[25] E. S. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson. Non-interactive key exchange. In *PKC 2013*, volume 7778, pages 254–271. Springer, 2013.

[26] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: validating SSL certificates in non-browser software. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, pages 38–49, New York, NY, USA, 2012. ACM.

[27] J. Gettys and K. Nichols. Bufferbloat: dark buffers in the internet. *Commun. ACM*, 55(1):57–65, Jan. 2012.

[28] P. K. Gummadi, S. Saroiu, and S. D. Gribble. King: estimating latency between arbitrary Internet end hosts. In *Proceedings of the 2nd Workshop on Internet Measurement*, pages 5–18, New York, NY, USA, 2002. ACM.

[29] A. Hiltgen, T. Kramp, and T. Weigold. Secure Internet banking authentication. *IEEE Security Privacy*, 4(2):21–29, March–April 2006.

[30] J. Ioannidis and S. M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proceedings of the 9th Network and Distributed System Security Symposium*, Reston, VA, USA, Feb. 2002. The Internet Society.

[31] T. Jaeger, K. Butler, D. H. King, S. Hallyn, J. Latten, and X. Zhang. Leveraging IPsec for mandatory access control across systems. In *Proceedings of the 2nd ACM conference on Computer and Communications Security*, New York, NY, USA, Aug. 2006. ACM.

[32] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In *Crypto 2012*, volume 7417, pages 273–293. Springer, 2012.

[33] A. Juels and J. G. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the 6th Network and Distributed System Security Symposium*, Reston, VA, USA, Feb. 1999. The Internet Society.

[34] C. Kaufman. RFC 4306: Internet key exchange (IKEv2) protocol, Dec. 2005. Status: PROPOSED STANDARD.

[35] A. D. Keromytis, S. Ioannidis, M. B. Greenwald, and J. M. Smith. The STRONGMAN architecture. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition*, volume 1, pages 178–188, 2003.

[36] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computing Systems*, 10(4):265–310, Nov. 1992.

[37] A. Langley. Transport Layer Security (TLS) Snap Start. Internet Engineering Task Force, June 2010. http://tools.ietf.org/html/draft-agl-tls-snapstart-00 (accessed Aug 26, 2013).

[38] A. Langley. Forward secrecy for Google HTTPS, Nov. 2011. https://www.imperialviolet.org/2011/11/22/forwardsecret.html (accessed Aug 26, 2013).

[39] A. Langley. How to botch TLS forward secrecy, June 2013. https://www.imperialviolet.org/2013/06/27/botchingpfs.html (accessed Aug 26, 2013).

[40] A. Langley, N. Modadugu, and W.-T. Chang. Overclocking SSL. In *Velocity: Web Performance and Operations Conference*, Santa Clara, CA, June 2010. http://www.imperialviolet.org/2010/06/25/overclocking-ssl.html (accessed Aug 26, 2013).

[41] A. Langley, N. Modadugu, and B. Moeller. Transport Layer Security (TLS) False Start. Internet Engineering Task Force, June 2010. http://tools.ietf.org/html/draft-bmoeller-tls-falsestart-00 (accessed Aug 26, 2013).

[42] E. Le Malécot, Y. Hori, and K. Sakurai. Preliminary insight into distributed SSH brute force attacks. *Proceedings of the IEICE General Conference*, page 2, Mar. 2008.

[43] M. Liberatore and B. N. Levine. Inferring the source of encrypted HTTP connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 255–263, New York, NY, USA, Oct. 2006. ACM.

[44] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the Linux operating system. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 29–42, Berkeley, CA, June 2001. The USENIX Association.

[45] D. McGrew. RFC 5116: An interface and algorithms for authenticated encryption, 2008. Status: PROPOSED STANDARD.

[46] W. M. Petullo and J. A. Solworth. Authentication in Ethos. https://www.ethos-os.org/papers/, June 2013.

[47] W. M. Petullo and J. A. Solworth. Simple-to-use, secure-by-design networking in Ethos. In *Proceedings of the Sixth European Workshop on System Security*, New York, NY, USA, Apr. 2013. ACM. https://www.ethos-os.org/papers/.

[48] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan. TCP fast open. In *Proceedings of the 7th International Conference on Emerging Networking Experiments and Technologies*, New York, NY, USA, 2011. ACM.

[49] E. Rescorla and N. Modadugu. RFC 6347: Datagram transport layer security version 1.2, 2012. Status: PROPOSED STANDARD.

[50] R. L. Rivest and B. Lampson. SDSI — a simple distributed security infrastucture. Technical report, MIT, Apr. 1996.

[51] S. Schillace. Default HTTPS access for Gmail, Jan. 2010. http://gmailblog.blogspot.com/2010/01/default-https-access-for-gmail.html (accessed Aug 26, 2013).

[52] J. A. Solworth. The Ethos operating system. http://www.ethos-os.org.

[53] J. A. Solworth and W. Fei. sayI: Trusted user authentication at Internet scale. https://www.ethos-os.org/papers/, Aug. 2013.

[54] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Proceedings of the 10th USENIX Security Symposium*, Berkeley, CA, USA, Aug. 2001. USENIX Association.

[55] S. Souders. Velocity and the bottom line. O'Reilly Media, July 2009. http://programming.oreilly.com/2009/07/velocity-making-your-site-fast.html (accessed Aug 26, 2013).

[56] E. Stark, L.-S. Huang, D. Israni, C. Jackson, and D. Boneh. The case for prefetching and prevalidating TLS server certificates. In *Proceedings of the 19th Network and Distributed System Security Symposium*, Reston, VA, USA, 2012. The Internet Society.

[57] R. Stewart. RFC 4960: Stream Control Transmission Protocol, Sept. 2007. Status: PROPOSED STANDARD.

[58] N. Vratonjic, J. Freudiger, V. Bindschaedler, and J.-P. Hubaux. The inconvenient truth about web certificates. In *Proceedings of the 10th Workshop on the Economics of Information Security*, June 2011.

[59] N. Weaver, R. Sommer, and V. Paxson. Detecting forged TCP reset packets. In *Proceedings of the 16th Network and Distributed Systems Security Symposium*, Reston, VA, USA, Feb. 2009. The Internet Society.

[60] J. E. White. A high-level framework for network-based resource sharing. In *Proceedings of the 1976 National Computer Conference and Exposition*, pages 561–570, New York, NY, USA, 1976. ACM.

[61] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos operating system. In *Proceedings of the 14th Symposium on Operating System Principles*, pages 256–269, New York, NY, USA, 1993. ACM.